Un langage imperatif de programmation au niveau tache : definition en logique temporelle

Eric Rutten, Lionel Marce

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

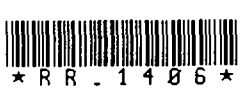
N° 1406

Programme 4
Robotique, Image et Vision

UN LANGAGE IMPERATIF DE PROGRAMMATION AU NIVEAU TACHE : DEFINITION EN LOGIQUE TEMPORELLE

Eric RUTTEN
Lionel MARCÉ

Avril 1991



UN LANGAGE IMPÉRATIF DE PROGRAMMATION AU NIVEAU TÂCHE : DÉFINITION EN LOGIQUE TEMPORELLE [1]

Eric RUTTEN[2]
Projet PRISME
INRIA / ERCIM
2004, route des Lucioles
06560 Valbonne Cedex
tél. 93 95 75 17
rutten@carolus.cma.fr

Lionel MARCÉ
Université de Bretagne Occidentale
Département Informatique
6, av. Victor Le Gorgeu
29283 Brest Cedex
tél. 98 31 63 89
marce@ubolib.cicb.fr

Résumé

L'exécution de plans d'actions par un robot a fait l'objet de diverses représentations, du point de vue de la planification en intelligence artificielle comme du contrôle d'exécution. Les formalismes de planification sont souvent fondés sur une logique des prédicats du premier ordre, ou plus récemment sur des logiques non-classiques, notamment temporelles. Les langages de programmation et de contrôle des robots, quand ils concernent le niveau tâche qui nous intéresse, comprennent des structures de contrôle des langages de programmation classiques, ainsi que d'autres plus spécifiquement liés à l'aspect temps-réel de leur exécution.

Nous proposons un modèle logique et temporel de plans d'actions dotés d'une structure de contrôle impérative. Dans ce but, nous définissons, sur la base d'une logique temporelle à intervalles, un ensemble de primitives de contrôle qui spécifient l'arrangement temporel des actions et sous-plans qu'elles encadrent. Ensuite, des primitives sont définies dans le même formalisme pour représenter la réaction aux évolutions dans l'environnement d'exécution du plan : de cette façon, on répond aux exigences d'interaction et d'adaptation au monde extérieur de notre domaine d'application, la robotique.

AN IMPERATIVE LANGUAGE FOR TASK-LEVEL PROGRAMMING: DEFINITION IN TEMPORAL LOGICS

Abstract

The execution of plans of actions by a robot inspired the conception of various representations, from the point of view of planning in artificial intelligence and of execution control. Planning formalisms are often founded on predicate logic and its more recent extensions, particularly temporal. Robot programming languages, when concerning the task-level, feature control structures from computer programming languages, as well as more specifically real-time oriented ones.

We propose a logical and temporal model of plans of actions provided with an imperative control structure. We therefore define, on the basis of an interval-based temporal logic, a set of imperative control primitives that define the temporal arrangement of the actions and subplans within their scope. Then, primitives for the reaction to evolutions in the environment are defined in the same formalism, in order to respond to constraints concerning interaction and adaptation to the external world in our application domain: robotics.

---

[1] Ce travail a été effectué à l'IRISA/INRIA-Rennes, 35042 RENNES Cedex

[2] boursier post-doctoral ERCIM (European Research Consortium in Informatics and Mathematics, grouping CNR (Italy), CWI (The Netherlands), GMD (Germany), INESC (Portugal), INRIA (France), RAL (United-Kingdom))

1 Introduction

1.1 Contexte et motivations

Le contexte général des travaux que nous présentons ici est celui de la conception d'environnements de programmation de systèmes contrôlant des processus physiques, et plus particulierement de systèmes robotiques étudiés dans le Projet de Robotique a l'IRISA/INRIA-Rennes. La motivation particulière qui nous guide est l'application de tels concepts à la programmation de robots au niveau tâche, c'est-à-dire au niveau concerné par l'enchaînement de tâches considérées comme primitives. Les concepts de tâches et de leur arrangement temporel pour former un plan sont toutefois plus généraux et trouvent un écho aussi bien dans la communication homme-machine que dans la génération de plans en intelligence artificielle.

Dans le domaine particulier de la robotique qui nous intéresse, comme dans d'autres, existent deux approches complémentaires : la robotique autonome (ou de substitution) et la robotique de coopération. En ce qui concerne la robotique autonome, où le robot est doté de facultés de décision suffisantes pour fonctionner sans besoin d'intervention humaine, le problème de la détermination de la suite des tâches à effectuer fait l'objet de travaux dans le domaine de génération de plans, ou planification, en intelligence artificielle. Une suite de tâches est alors appelée un plan, où chacune des tâches primitives est appelée une action. Comme nous le verrons plus loin, la planification fait intervenir des modèles reposant sur la représentation des connaissances en logique.

En robotique de coopération, il y a partage des facultés de décision, comme de perception, action ou communication, avec un opérateur humain. La planification peut alors y être vue comme une forme de programmation à haut niveau d'abstraction, c'est-à-dire au niveau tâche, où l'on considère les conditions logiques requises et les effets logiques produits par une action du robot. Le plan est alors un programme, définissant une structure de contrôle sur un ensemble d'actions qui sont les intructions primitives du langage.

Dans cette dernière approche, qui est celle qui nous intéresse, il faut donc un langage permettant l'expression de l'arrangement dans le temps des actions, sous forme d'un flot de contrôle qui guide le "séquencement" de ces tâches primitives. La définition de ce langage doit aussi se faire de façon à permettre d'y associer des fonctionnalités d'aide à la planification, et à servir de représentation de référence pour la mise en œuvre de l'exécution des plans d'actions.

Dans ces conditions, un modèle de plans d'actions fondé sur la logique, selon une approche courante en planification, est utile ici aussi pour :

- définir le comportement, c'est-à-dire une sémantique du langage de planification, en relation avec son exécution et les problèmes de programmation temps-réel qui l'accompagnent;
- constituer le support d'une assistance à la planification, par exemple sous forme de simulation, ce qui a fait l'objet d'une application du travail présenté ici, ou bien dans la perspective de l'utilisation possible de la génération de plans.

1.2 Problématique

Dans ce cadre général, la problématique à laquelle nous nous attaquons consiste à construire un modèle logique et temporel de plans d'actions dotés d'une structure de contrôle impérative.

De façon à permettre l'écriture de plans d'actions, le travail consiste à :

- déterminer les primitives de ce langage de planification : nous faisons le choix d'un langage impératif, présentant des structures de contrôle classiques des langages de programmation, ainsi que d'autres moins classiques, plus directement liées aux aspects de réactivité et de temps réel des applications visées.
- définir le comportement de chacune de ces primitives, en considérant ses aspects logiques et temporels.

Le travail présenté ici est donc une combinaison, d'une part, de l'application d'un formalisme théorique de représentation des connaissances : la logique temporelle à intervalles d'Allen et son utilisation à la représentation des actions et d'un environnement changeant; d'autre part, des méthodes et des langages de programmation au niveau tâche en robotique.

1.3 Organisation de l'article

Dans la section suivante, nous verrons comment le problème de représentation des plans est traité dans la littérature, en tant que problème de planification et représentation des connaissances, et en tant que problème de commande de robots au niveau tâche.

Nous introduisons ensuite brièvement les éléments de base du modèle, utilisant la logique temporelle à intervalles. Les plans seront construits à partir de ceux-ci d'abord au moyen de structures de contrôle classiques, caractérisées par le fait qu'elles sont temporellement indépendantes de leur environnement d'exécution. Ensuite, le langage et son modèle sont étendus à la réactivité par des éléments permettant de lier l'exécution à des changements survenant dans l'environnement. Un exemple illustre ceci, en traitant une application à une mission d'un système téléopéré à deux bras manipulateurs, dans un environnement de télérobotique spatiale.

2 Représentations des plans d'actions

Nos intérêts correspondent donc à deux domaines dont les littératures sont abondantes; pour rester brefs, nous n'ébaucherons ici que les grandes lignes d'une étude plus poussée [Rutten 90].

2.1 Planification en intelligence artificielle

Le domaine de la génération de plans en intelligence artificielle a produit un important ensemble de résultats concernant les problèmes d'exploration d'espaces d'états liés au processus de génération lui-même [Hendler e.a. 90]. Nous nous intéressons plus aux représentations des opérateurs ou actions primitives, qui sont considérées comme étant directement exécutables par un agent. Cet agent est généralement décrit comme étant un robot, regardé toutefois d'un point de vue très abstrait.

La représentation la plus classique est celle de *STRIPS* : elle définit une action en termes de manipulations d'une base de données constituée de prédicats de la logique du premier ordre. Une action y est caractérisée par trois listes de prédicats : les préconditions, qui doivent être présentes dans l'état courant de l'environnement, les ajouts, qui représentent ce qui devient vrai par effet de l'action, et les retraits, qui représentent ce qui devient faux. Depuis lors, de nombreux planificateurs ont repris ce modèle.

Les plans générés eux-mêmes sont des "ensembles organisés d'opérateurs", c'est-à-dire par exemple des ensembles d'actions munis d'un ordre total (plans linéaires) ou partiel (plans non-linéaires). La forme que doit prendre cette organisation, bien que faisant l'objet de nombreux travaux, ne donne pas lieu à un standard généralement accepté [DARPA 87]; celles proposées répondent en général à des considérations relevant du processus de génération, plutôt que de l'utilisation par ailleurs du plan produit.

2.2 Représentations temporelles

Une des approches récentes dans ce domaine a été d'étendre ce type de représentation en lui donnant une dimension temporelle, en remplaçant la logique classique utilisée comme formalisme de représentation par une logique temporelle, notamment la logique temporelle à intervalles, telle que l'a introduite Allen [Allen 81]. On peut remarquer par ailleurs que des logiques temporelles ont été utilisées à la spécification des systèmes, des programmes et de leur exécution [Dubois e.a. 91,Audureau e.a. 87] : il s'agit alors généralement de formes modales de logique temporelle [Bestougeff e.a. 89]. Se donner le moyen de s'exprimer explicitement à propos du temps permet de représenter les changements dans l'environnement, dûs aux actions ou non, aussi bien que la disposition temporelle relative de ces actions.

Nous présentons ici brièvement le modèle d'Allen : les intervalles y sont des "morceaux" ("*chunks*") de temps, ayant une étendue. Des relations de position sont définies : un intervalle i peut être placé *avant (b), égaler (=), rencontrer (m), chevaucher (o),* être *pendant (d), commencer (s)* ou *terminer (f)* un autre intervalle j, ou bien dans une relation réciproque. Ces treize relations primitives, mutuellement exclusives, décrivent tous les positionnements relatifs possibles pour une paire d'intervalles. Ces relations peuvent être groupées en relations disjonctives si la relation précise n'est pas connue : $((I_1\ r_1\ I_2) \vee (I_1\ r_2\ I_2))$ peut être écrit : $(I_1\ (r_1\ r_2)\ I_2)$; par exemple, la relation $I_1\ (d\ s\ o)\ I_2$.

On peut alors définir, pour disposer d'une contenance non-stricte, deux relations :

$$I_1\ dur\ I_2 \equiv I_1\ (s\ d\ f)\ I_2, \text{ et } I_1\ con\ I_2 \equiv I_1\ (si\ di\ fi)\ I_2\ .$$

Par ailleurs, on peut noter ? la disjonction des treize relations :

$$I_1 \ ?\ I_2 \equiv I_1 \ (b\ a\ d\ di\ o\ oi\ m\ mi\ s\ si\ f\ fi\ =)\ I_2$$

Sur ces bases, une table de transitivité a été construite [Allen 83], permettant de déterminer, connaissant deux relations r_1 et r_2 entre trois intervalles I_1, I_2 et I_3, c'est-à-dire $I_1\ r_1\ I_2$ et $I_2\ r_2\ I_3$, une troisième relation r_3 obtenue transitivement : $I_1\ r_3\ I_3$.

Dans cette logique, la vérité d'une proposition P sur un intervalle de temps I est notée : $vrai(I, p)$. Cette représentation a donné lieu à de nombreux développements, dont ceux dûs à Shoham [Shoham 86]. Les techniques permettant de mettre en œuvre efficacement un raisonnement temporel ont été étudiées en profondeur, notamment en ce qui concerne l'algorithmique associée et sa complexité [Bestougeff e.a. 89, Ghallab 89].

Une représentation de ce type permet de représenter les actions par leurs aspects logiques, comme dans *STRIPS*, en leur donnant en plus une dimension temporelle nécessaire pour exprimer la durée des actions ou leurs arrangements complexes. Sandewall e.a. [Sandewall e.a. 86] et Alami e.a. [Alami e.a. 90] ont proposé des représentations déclaratives de plans d'actions, en termes de relations entre leurs intervalles d'exécution, ou les instants d'extrémités de ces intervalles.

La seule représentation, à notre connaissance, qui soit à la fois temporelle et impérative est proposée par Hultman e.a. [Hultman e.a. 89]. Elle présente des définitions temporelles de l'action, de la séquence, du parallélisme, de la conditionnelle, de l'itération et d'une forme de réactivité. Ce modèle est développé sur les bases de l'approche de Sandewall e.a. [Sandewall e.a. 86]. Ces travaux, ainsi que ceux de Ghallab e.a. [Alami e.a. 90] par exemple, étaient liés au développement d'un robot mobile et de son environnement de programmation.

Ceci nous mène au lien entre un tel modèle de description logique et temporelle, et les langages de commande et programmation exécutables par des robots.

2.3 Robotique et langages

De ce point de vue, la détermination d'un plan d'action pour un robot peut être vue comme un problème de programmation, caractérisé par un environnement évolutif, des objets particuliers (mouvements, contrôle au sens de l'automatique, géométrie), et un contrôle temps-réel très lié aux problèmes de la réactivité. Comme le soulignent Sandewall e.a. [Sandewall e.a. 86], il existe un rapport entre l'exécution d'un programme sur un ordinateur et celle d'un plan par un système robotisé : il transparaît dans la parenté entre les langages de programmation et les exemples de langages de commande à haut niveau que l'on peut trouver en robotique.

Une hiérarchie classique présentée par Gaspart [Gaspart 87] classifie les langages en programmation robotique de niveau articulation, de niveau effecteur (par exemple un bras manipulateur), de niveau objet (en termes des objets manipulés) et de niveau objectif (spécification du but final). La programmation des robots est le plus souvent étudiée aux niveaux d'abstraction les plus bas, s'intéressant à des problèmes de mouvement géométriques ou de traitement d'information capteurs. Le niveau qui nous intéresse est le niveau tâche où, comme le disent Schroeder-Trovato e.a., les langages de commande sont aux instructions de programmation robotique ce que les langages de programmation à haut niveau sont à l'assembleur [Schroeder-Trovato e.a. 87]. Il existe diverses approches à ce problème [Gravez 88], mais il est difficile de trouver une approche générale concernant ce niveau d'abstraction [Voltz 88] ; toutefois, plusieurs solutions spécifiques [Schroeder-Trovato e.a. 87,Michalowski e.a. 87,Hultman e.a. 88] présentent des aspects communs.

Ces langages sont exécutés sur des architectures de robots hiérarchiques, et présentent des structures de contrôle classiques, telles que la séquence, la conditionnelle et différentes interprétations du parallélisme; celles-ci sont souvent complétées par des structures de contrôle spécifiant que l'exécution d'une action ou d'un plan doit avoir lieu dès qu'une condition est satisfaite dans l'environnement extérieur. Ces structures de contrôle ont parfois des similitudes avec certaines utilisées pour la représentation des tâches en communication homme-machine [Scapin e.a. 89].

Dans le domaine des langages de programmation, les structures de contrôle classiques que sont la séquence, la conditionnelle et l'itération ont été étendues par les commandes gardées et le parallélisme, dont les différentes interprétations peuvent être vues comme autant de façons de disposer des instructions dans le temps [Filman e.a. 84,Andrews e.a. 83]. D'autres approches concernent les automates et les réseaux de Petri [Paoletti e.a. 89].

Dans le domaine de la programmation temps-réel enfin, des langages ont été conçus avec une préoccupations de bon fondement dans la théorie des systèmes réactifs : parmi les langages temps-réel synchrones (ESTEREL [Berry e.a. 87], LUSTRE [Caspi e.a. 87], SIGNAL [Benveniste e.a. 89]), ESTEREL est impératif, et présente des primitives de contrôle spécifiquement temporelles.

En conclusion, les représentations des plans d'actions que nous venons d'exposer seront une source d'inspiration pour le modèle que nous proposons : celui-ci consiste en la définition d'éléments d'un langage impératif dans les mêmes termes de logique temporelle à intervalles qui permettent de modéliser à la fois l'environnement et les actions. De cette façon, nous tentons de combiner la planification, et particulièrement ses représentations temporelles, avec la programmation de robots au niveau tâche, particulièrement dans ses aspects impératifs et de temps-réel.

Une présentation complète en est faite ailleurs [Rutten 90] ; nous nous concentrerons ici sur les définitions logiques des primitives de contrôle proposées.

3 Eléments de base

3.1 Logique temporelle à intervalles

La définition des faits temporels est inspirée des travaux d'Allen. La primitive de base qui sera utilisée dans la suite est l'intervalle de temps. Elle s'interprète de la façon habituelle : il s'agit d'un morceau de temps qui a une étendue, ou durée, et qui est convexe. Ceci donne lieu aux résultats présentés en section 2.2. On définit l'inclusion non-stricte, à la différence de *d*, désignée par la relation *dans* par :

$$I_1 \ dans \ I_2 \equiv I_1 \ (s \ f \ d \ =) \ I_2$$

On aura besoin de se donner un prédicat *dure(I, D)*, associant une durée quantitative D à un intervalle I.

Les propriétés du monde seront représentées sous la forme générale :

$$vrai(Intervalle, Propriété)$$

La signification intuitive en est que la *Propriété* est vraie sur l'étendue de *Intervalle*. Leur interprétation est celle classique dans la littérature, exposée en section 2.2. En particulier, la négation sera dotée de l'interprétation dite "forte", c'est-à-dire que *non(P)* sera vraie sur I si P n'est vraie sur aucun sous-intervalle de I :

$$vrai(I, non(P)) \Longleftrightarrow (\forall i)(\ i dans I \Longrightarrow \neg vrai(i, P)\ )$$

ce qui la différencie de la négation faible, pour laquelle il suffit qu'il existe un sous-intervalle de I sur lequel P ne soit pas vraie.

L'exécution des actions et des plans sur un intervalle est représentée par :

$$exec(Intervalle, Action - ou - Plan)$$

Sa signification est que l'action ou le plan est exécuté du début à la fin sur l'intervalle I.

3.2 Les actions primitives

Le niveau tâche Nous nous intéressons au niveau tâche décrit en section 2.3. Une action primitive, ou tâche, y est considérée du point de vue de ses conditions et effets logiques sur l'environnement, et de sa dimension temporelle.

On la définira par :

- son intervalle temporel d'exécution,
- les relations de celui-ci aux intervalles de vérité des faits temporels concernés.

Représentation temporelle La définition temporelle d'une action primitive sera inspirée des représentations classiques de l'action comprenant préconditions, retraits et ajouts, en termes de prédicats de la logique du premier ordre dans la représentation du monde. Une dimension temporelle y sera introduite, ainsi que des simplifications et variations par rapport à des propositions antérieures [Rutten e.a. 89*b*], par unification de la représentation des effets et distinction entre les préconditions, nécessaires avant l'exécution, et les conditions nécessaires pendant celle-ci.

Une action primitive est un quintuplet, noté sous forme de terme :

$$action(\ A,\ Durée,\ Préconditions,\ Conditions,\ Effets\ ).$$

La façon dont cette action s'inscrit dans le temps est illustrée par la fig 1.

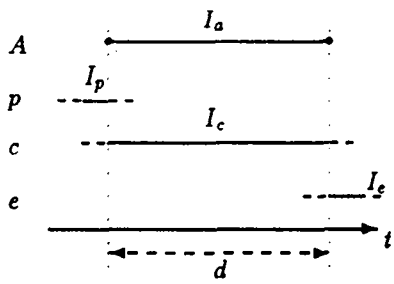


Figure 1: Une action primitive dans le temps.

Chaque précondition p de l'ensemble P doit être vérifiée au moins immédiatement avant l'exécution de l'action, c'est-à-dire sur un intervalle I_p tel que :

$$I_p\ (m\ o\ fi\ di)\ I_a$$

Chaque condition c de l'ensemble C doit être vérifiée sur tout l'intervalle de l'action I ; c'est-à-dire qu'elle doit être associée à un intervalle de vérité I_c, tel que I_a soit *dans* I_c :

$$I_a\ (s\ f\ d\ =)\ I_c, \text{ c'est-à-dire : } I_c\ (si\ fi\ di\ =)\ I_a$$

Quant aux effets, la véritable contrainte qu'ils doivent vérifier est qu'ils doivent être satisfaits immédiatement après l'exécution de l'action ; ils peuvent néanmoins l'être déjà avant la fin de celle-ci (par exemple les actions d'ouvrir une porte déjà ouverte, ou de fermer une porte qui n'est pas ouverte, ne doivent pas donner lieu à un échec, ou alors la modélisation de cet aspect de l'action doit se faire sous la forme d'une condition explicite). La disposition relative des intervalles de l'action : I et d'un effet e : I_e, est alors :

$$I_e\ (di\ si\ oi\ mi)\ I_a \text{ ou : } I_a\ (d\ s\ o\ m)\ I_e$$

Une spécification de la disposition temporelle d'une telle action A, définie par *action(A, D, P, C, E)*, peut être donnée comme suit :

Définition 1 :

$$\begin{aligned} \text{exec}(I_a,\text{A}) \iff\ & \text{dure}(I_a, D) \\ & \wedge(\forall p \in P)(\exists I_p)(\ \text{vrai}(I_p, p) \wedge I_p\ (m\ o\ fi\ di)\ I_a) \\ & \wedge(\forall c \in C)(\exists I_c)(\ \text{vrai}(I_c, c) \wedge I_a \text{ dans } I_c) \\ & \wedge(\forall e \in E)(\exists I_e)(\ \text{vrai}(I_e, e) \wedge I_a\ (d\ s\ o\ m)\ I_e) \end{aligned}$$

Les conditions de l'ensemble C sont une composante importante de la dimension temporelle de cette définition, ce qui la différencie des modèles non-temporels, dans le sens où elles mettent en relation l'étendue de l'intervalle de l'action avec des faits temporels dans l'environnement. Elles permettent de s'exprimer au sujet d'actions qui consistent à maintenir une condition dans l'environnement pendant leur

exécution, c'est-à-dire dont l'exécution doit garantir qu'elle restera satisfaite. Ceci correspond à ce que Sandewall e.a. [Sandewall e.a. 86] appellent les "*keep-actions*", dont les "*prevail-conditions*" correspondent aux conditions présentées ici.

Un exemple particulier d'action primitive est l'action *rien* qui est sans durée, sans condition et sans effet : *action(rien, 0, [ ], [ ], [ ])*.

Actions à durée indéfinie Dans le domaine d'application considéré : la robotique d'intervention, la durée des actions n'est pas toujours connue à l'avance. Il y a des actions dont la durée peut changer d'une exécution à l'autre et on ne veut pas la spécifier au moment où l'on donne la définition générale. Cette durée doit alors être déterminée à l'exécution : dans notre modèle, cela signifie que l'intervalle va être déterminé par la structure de contrôle englobante.

Une action à durée indéfinie a, hormis en ce qui concerne la durée, les mêmes caractéristiques qu'une action primitive telle que celles vues plus haut. Elle est notée :

action-d-i(A, Préconditions, Conditions, Effets).

et est définie temporellement, pour une action à durée indéfinie A définie par *action-d-i(A, P, C, E)*, par :

Définition 2 :

$$\begin{aligned} \text{exec(I,A)} \iff\ & (\forall p \in P)(\exists I_p)(\ \text{vrai}(I_p, p) \wedge I_p\ (m\ o\ fi\ di)\ I) \\ & \wedge(\forall c \in C)(\exists I_c)(\ \text{vrai}(I_c, c) \wedge I \text{ dans } I_c \\ & \wedge(\forall e \in E)(\exists I_e)(\ \text{vrai}(I_e, e) \wedge I\ (d\ s\ o\ m)\ I_e) \end{aligned}$$

La différence avec l'action primitive de la définition 1 est que la durée de l'intervalle d'exécution n'est pas contrainte à la déclaration de l'action, mais à l'utilisation. Un exemple particulier d'action à durée indéfinie est *pause* : *action-d-i(pause, [ ], [ ], [ ])*.

4 Primitives Indépendantes de l'Environnement

Ces primitives sont temporellement indépendantes de l'environnement, de par le fait qu'elles se définissent en termes de leurs composants. Un plan est considéré ici comme un ensemble d'actions muni d'une *structure de contrôle*. Un plan est alors soit une action seule (*action primitive*), soit composé de sous-plans, qui sont des plans eux-mêmes récursivement, encadrés par une structure de contrôle.

4.1 Structures de contrôle classiques

Pour chacune on donnera une définition temporelle, qui spécifie formellement sa signification intuitive. Des variantes peuvent être trouvées à ces définitions [Rutten 90].

4.1.1 Séquence

Intuitivement, il s'agit simplement de spécifier que les différents sous-plans de la séquence ont lieu l'un après l'autre, c'est-à-dire que chaque sous-plan ne commence que quand le précédent est terminé.

Elle est notée :

seq(ListeP),

où la liste de sous-plans *ListeP* a pour forme $[P_1|P]$, P_1 étant le premier sous-plan dans la liste, et P le reste de la liste. Cet opérateur de contrôle indique que les sous-plans de la séquence sont exécutés l'un après l'autre, dans l'ordre de la liste, à l'intérieur d'un intervalle global I :

Définition 3 :

$$\begin{aligned} \text{exec}(I,\text{seq}([P_1|P])) \iff (\exists I_1, I_P)\quad & (I_1\ s\ I\ \wedge\ I_1\ m\ I_P \wedge I_P\ f\ I \\ & \wedge\ exec(I_1, P_1)\ \wedge\ exec(I_P, \text{seq}(P))\ ). \end{aligned}$$

Le cas où la liste de sous-plans est vide, est assimilé à l'exécution de l'action *rien*.

4.1.2 Conditionnelle

Intuitivement, on veut pouvoir spécifier que, suivant qu'une condition est satisfaite ou non dans l'environnement d'exécution, on exécute un plan ou bien un autre.

Elle est notée:

$$cond(C, P_{vrai}, P_{faux}).$$

La condition C est évaluée au début de l'intervalle associé à la structure, et selon le résultat, le sous-plan correspondant est choisi pour l'exécution, l'autre étant abandonné :

Définition 4 :

$$\begin{aligned}\text{exec}(I, \text{cond}(C, P_{vrai}, P_{faux})) \Longleftrightarrow (\exists I_c)(\quad & I_c \; s \; I \wedge \\ & ((\text{vrai}(I_c, C) \wedge \text{exec}(I, P_{vrai})) \vee \\ & (\text{vrai}(I_c, non(C)) \wedge \text{exec}(I, P_{faux}))) \\ ).\end{aligned}$$

4.1.3 Parallélisme

Il s'agit de spécifier que plusieurs sous-plans ont lieu "en même temps", c'est-à-dire sur une même période de temps. Il serait toutefois restrictif de leur imposer un intervalle d'exécution identique, comme le font par exemple Hultman e.a. [Hultman e.a. 89] : en effet les différents plans exécutés en parallèle n'ont pas nécessairement la même durée.

Les motivations de l'utilisation du parallélisme se trouvent en partie dans le gain de temps, à l'exécution, apporté par la simultanéité de plusieurs actions. Elles se trouvent aussi dans le nécessité de pouvoir s'exprimer au sujet du parallélisme de l'architecture des robots considérés, qui sont constitués de différents modules simultanément actifs [Jauffrineau 88,Paoletti 91].

Dans la perspective choisie de clarté de la structuration, et d'imbrication hiérarchique des primitives de contrôle, une forme du parallélisme adéquate est l'adaptation du **cobegin-coend**. Le constructeur parallèle est noté

$$par(B),$$

où l'ensemble de sous-plans B contient les plans constituant chacun une branche parallèle aux autres. Dans cette structure, toutes les branches b de la liste B commencent ensemble, et le parallélisme termine quand toutes les branches ont terminé, c'est-à-dire avec la branche la plus longue :

Définition 5 :

$$\text{exec}(I, par(B)) \Longleftrightarrow (\exists b_1 \in B) \; (\; \text{exec} \; (I, b_1) \;) \quad \wedge (\forall b_2 \in B)(\exists I')(\text{exec}(I', b_2) \wedge \; I' \; (s \; =) \; I).$$

4.2 Intégration des relations de Allen

Les structures précédemment présentées sont des particularisations de la disposition temporelle des sous-plans qu'elles encadrent. Dans ce sens, elles sont une restriction des relations générales d'Allen. Par exemple, la séquence correspond (définition 3) à la relation de rencontre : m, appliquée aux sous-plans qui la composent. La relation entre les branches d'une structure parallèle, dans sa définition 5, est (s *si* $=$).

Relation entre sous-plans On peut donc se donner le moyen, réciproquement, de définir des primitives de contrôle reprenant ces relations générales, et spécifiant qu'entre les intervalles d'exécution de deux sous-plans encadrés par cette primitive, on aura une certaine relation.

On définit par exemple le chevauchement o :

Définition 6 :

$$\begin{aligned}\text{exec}(I, o(Plan_1, Plan_2)) \Longleftrightarrow \quad & \text{exec}(I_1, Plan_1) \wedge \text{exec}(I_2, Plan_2) \\ & \wedge I_1 \; o \; I_2 \\ & \wedge I_1 \; (s \; f \; =) \; I \wedge I_2 \; (s \; f \; =) \; I\end{aligned}$$

De façon générale, pour une relation r :

Définition 7 :

$$\begin{aligned} \text{exec}(I, r(Plan_1, Plan_2)) \Longleftrightarrow \quad & \text{exec}(I_1, Plan_1) \wedge \text{exec}(I_2, Plan_2) \\ & \wedge I_1\ r\ I_2 \\ & \wedge I_1\ (s\ f\ =)\ I \wedge I_2\ (s\ f\ =)\ I \end{aligned}$$

Cas de la relation *rencontre* (m) Un cas intéressant est celui de la relation de *rencontre* m : il se ramène exactement à la définition 3 de la séquence. En effet, par définition, pour $m(Plan_1, Plan_2)$ on a :

$$\begin{aligned} exec(I, m(Plan_1, Plan_2)) \Longleftrightarrow \quad & exec(I_1, Plan_1) \wedge exec(I_2, Plan_2) \\ & \wedge I_1\ m\ I_2 \\ & \wedge I_1\ (s\ f\ =)\ I \wedge I_2\ (s\ f\ =)\ I \end{aligned}$$

c'est-à-dire, en simplifiant l'expression des relations entre les intervalles au moyen des tables transitivité, que de : $I_1\ m\ I_2 \wedge I_2\ (s\ f\ =)\ I$ on déduit : $I_1\ (m\ d\ s\ o)\ I$ et de la conjonction avec $I_1\ (s\ f\ =)\ I$, on déduit : $I_1\ s\ I$ et d'autre part, de : $I_1\ m\ I_2 \wedge I_1\ (s\ f\ =)\ I$ on déduit : $I_2\ (d\ f\ oi\ mi)\ I$ et de la conjonction avec $I_2\ (s\ f\ =)\ I$, on déduit : $I_1\ f\ I$.

On retrouve bien la séquence telle qu'elle a été posée par la définition 3 :

$$\begin{aligned} exec(I, m(Plan_1, Plan_2)) \Longleftrightarrow \quad & exec(I_1, Plan_1) \wedge exec(I_2, Plan_2) \\ & \wedge I_1\ m\ I_2 \\ & \wedge I_1\ s\ I \wedge I_2\ f\ I \end{aligned}$$

Relation à un intervalle D'une façon différente, on peut définir des primitives s'exécutant sur un intervalle I et spécifiant qu'un sous-plan P doit être exécuté sur un intervalle I_P ayant une certaine relation avec I.

Par exemple la primitive $di(P)$ spécifie que le plan P doit être exécuté sur un intervalle I_P *pendant* l'intervalle I : $I\ di\ I_P$, d'une façon qui n'est pas particulièrement déterminée. Elle est définie par :

Définition 8 :

$$\text{exec}(I, di(P)) \Longleftrightarrow \text{exec}(I_P, P) \wedge I\ di\ I_P$$

4.3 Abstraction

Nous avons un premier ensemble de constructeurs de base, qui sont définis pour être assemblés de façon à former des plans. A cette fin, il peut être utile d'avoir la possibilité de définir de nouveaux éléments à partir de ceux de base, et de les réutiliser. Ceci peut être fait ici de diverses manières.

Abstraction de sous-plans Pour permettre de nommer et réutiliser des plans, on se donne la notation par un couple :

$$action\text{-}c(A,\ P),$$

où l'action composée A est définie par le plan P, et qui s'interprète comme suit :

Définition 9 :

$$\text{exec(I, A)} \Longleftrightarrow \text{action-c(A, P)} \wedge \text{exec(I, P)}$$

Décomposition d'action De la même façon, des actions plus complexes peuvent être définies en utilisant un plan. Ceci permet de considérer des actions aux effets répartis le long de leur durée, ou dépendant, par une conditionnelle, du contexte dans lequel elles sont exécutées. Par exemple, une façon de définir une action a ayant des effets dans le courant de son intervalle, comme l'illustre la fig. 2, est de la décomposer en actions primitives, de la manière suivante : $action\text{-}c(\ a\ ,\ seq([a_1, a_2])\ )$, où l'action a_1 est telle que : $action(a_1,\ d_1,\ [\ ],\ [p_1,\ p_2],\ [non(p_2),\ p_3])$, et a_2: $action(\ a_2,\ d_2,\ [p_1,\ p_2],\ [p_3],\ [p_2]\ )$.

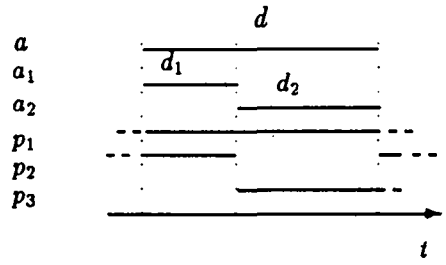


Figure 2: Une action composée dans le temps: *action-c(a,seq([a_1,a_2]))*.

Définition de structures de contrôle dérivées A partir de ces opérateurs, d'autres, plus spécifiques, peuvent être construits, comme, par exemple, l'*itération conditionnelle*, définie récursivement par :

action-c (tant-que(C, corps *),*
*cond(*C, *seq([* corps,
*tant-que(*C, corps*)]),*
rien))

On peut définir une conditionnelle simple : si la condition C est satisfaite, alors le sous-plan P est exécuté, sinon *rien* :

action-c (si(C,P), cond(C, P, rien))

4.4 Structures de contrôle associées aux actions à durée indéfinie

Jusqu'ici, la définition temporelle des structures se faisait en fonction de celle des sous-structures. Maintenant, nous allons définir des actions et des structures telles que l'intervalle d'une sous-structure sera défini en fonction de celui de la structure englobante, elle-même définie en fonction d'autres sous-structures.

Une façon de déterminer l'intervalle d'exécution d'une action à durée indéfinie est de l'associer à l'exécution d'un plan en parallèle. L'action A commence alors en même temps que le plan P, et ne termine que quand celui-ci termine. On peut noter ceci :

aussi-longtemps-que(P,A) ou : *a-l-q(P,A)*

et le définir temporellement par :

Définition 10 :

exec(I,a-l-q(P,A)) $\Longleftrightarrow$ exec(I,P) $\wedge$ exec(I,A)

On peut étendre ce constructeur en permettant l'association à un plan d'un ensemble d'actions à durée indéfinie, par la primitive :

a-l-q-ens(Plan,EnsAct)

Ceci donne lieu à la définition temporelle :

Définition 11 :

exec(I,a-l-q-ens(P, EA)) $\Longleftrightarrow$ exec(I, P) $\wedge$ ($\forall$ A $\in$ EA)(exec(I, A))

Une autre manière de faire est de spécifier la disposition temporelle de ces actions à durée indéfinie de façon arbitraire, en donnant par exemple l'intervalle d'exécution I_a, dans la primitive :

faire-sur(I_a,A)

définie temporellement par :

Définition 12 :

exec(I,faire-sur(I_a, A)) $\Longleftrightarrow$ I_a (f =) $I \wedge$ exec(I_a, A)

On peut aussi imposer la durée D, par la primitive :

faire-durant(D,A)

définie temporellement par :

Définition 13 :

exec(I,faire-durant(D, A)) $\Longleftrightarrow$ *dure*(I, D)$\wedge$ exec(I, A)

A partir de cette dernière structure, on peut définir une action de délai :

action-c(délai(D), faire-durant(D,pause)).

Il n'y a jusqu'ici que peu d'interaction avec l'environnement extérieur : il s'agit des conditions d'exécution des actions et des variantes de la conditionnelle. Ceci est dû au choix que nous avons fait de nous concentrer d'abord sur les structures de contrôle indépendantes de l'environnement dans leur spécification temporelle. Or on a besoin, surtout dans la perspective d'une application en télérobotique, de pouvoir spécifier une *réaction à l'environnement*, à des changements qui y ont lieu et à des évolutions qui s'y déroulent. On doit pouvoir spécifier l'attente de la satisfaction de ces conditions, et la réaction immédiate à leur occurence.

5 Primitives de Réaction à l'Environnement

5.1 Extension de la problématique et du modèle

Là où les plans étaient définis dans les seuls termes de leurs composants et de leurs relations internes, il faut maintenant introduire la possibilité de s'exprimer à propos d'arrangements temporels de ces plans en relation avec l'environnement dans lequel ils s'exécutent, et en réaction à son évolution dans le temps, y compris les interventions de l'opérateur. De façon générale, on a besoin de pouvoir spécifier qu'une action ou un plan doit être exécuté "quand", ou "dès que", certaines conditions sont réunies dans l'environnement ou par l'opérateur.

Or, les primitives de contrôle classiques sont mal adaptées à la spécification de ce type de situations. Il faut donc se donner les moyens d'exprimer explicitement la dépendance temporelle d'un plan à des circonstances extérieures. Dans le domaine de la programmation réactive, la question a été considérée du point de vue des langages de programmation en temps-réel et de leur exécution. Certains concepts sont intéressants pour l'approche, plus descriptive, de modélisation présentée ici : il s'agit essentiellement d'aspects de la forme des primitives de contrôle. Les intervalles d'exécution de certaines actions, ou même de certains plans, doivent pouvoir être mis en relation avec les intervalles de vérité de faits temporels représentant les caractéristiques de l'environnement. On complètera le modèle temporel de plans par l'introduction de la notion de réaction au niveau des actions et au niveau des plans

5.2 Réaction au niveau des actions

Les actions à durée indéfinie vont être considérées dans le cadre de structures de contrôle qui les lieront à une condition qui déterminera (en partie) leur intervalle d'exécution, ainsi que celui de la structure concernée.

Une façon de spécifier l'exécution d'une action à durée indéfinie est de faire déterminer son intervalle par un fait temporel : l'action sera exécutée tant que la condition est satisfaite. La primitive est notée :

tant-que-condition(C,A) ou *tq-c(C,A)*

Son interprétation intuitive est que l'action A commence à s'exécuter avec la primitive *tq-c(C,A)*, et termine dès que la condition C cesse d'être vraie : c'est-à-dire que l'exécution de *tq-c(C,A)* correspond à l'exécution de l'action A sur un intervalle qui termine ou est égal à l'intervalle de vérité de C ; si C n'est pas vérifié, la primitive *tq-c(C,A)* s'exécute sur un intervalle de durée nulle, et l'action A n'est pas exécutée.

La définition temporelle correspondante est :

Définition 14 :

$$\begin{aligned} \text{exec(I, tq-c(C,A))} \Longleftrightarrow (\exists I_c, I_{nc})(\quad & \text{vrai}(I_c, C) \wedge I \ (f \ =) \ I_c \\ & \wedge \text{ vrai}(I_{nc}, non(C)) \wedge I \ m \ I_{nc} \\ & \wedge \text{ exec}(I, A) \ ) \\ \vee \ (\exists I_{nc})(\quad & \text{vrai}(I_{nc}, non(C)) \wedge I \text{ dans } I_{nc} \\ & \wedge \text{ dure}(I, 0) \ ) \end{aligned}$$

Une autre primitive, permettant d'exécuter l'action jusqu'à ce que la condition soit satisfaite, est notée :

jusqu'à-condition(C,A) ou *jq-c(C,A)*

Elle s'interprète comme l'exécution de A en attendant que la condition C soit satisfaite, c'est-à-dire que l'intervalle d'exécution de A rencontre celui de vérité de C. Il s'agit en fait d'une primitive symétrique à *tq-c(C,A)*, et on peut la définir en terme de cette dernière :

Définition 15 :

$exec(I, jq\text{-}c(C, A)) \Longleftrightarrow exec(I, tq\text{-}c(non(C), A))$

On peut construire à partir de ces primitives une primitive dérivée : l'attente de condition, définie par :

action-c(attente(C), jq-c(C,pause))

5.3 Réaction au niveau des plans : les règles

Il s'agit d'exprimer la réaction à la satisfaction d'une condition par l'exécution d'un plan : c'est-à-dire une *règle* spécifiant l'exécution de ce plan *dès que* la condition est satisfaite dans l'environnement ou par l'opérateur.

Ce point de vue est intermédiaire entre la planification réactive et la programmation de "règles de survie" sur un robot. En effet, la planification réactive se préoccupe de la façon de faire communiquer les processus de planification et d'exécution pour la détermination d'un nouveau plan en réaction soit à un changement dans l'environnement, soit à une impossibilité d'exécuter le plan courant. Or il s'agit ici de spécifier, dans le cadre même du plan, des réactions à certaines situations, et ceci dans des primitives intégrées dans la structure de contrôle globale, et d'autant mieux imbriquées qu'elles sont définies dans le même formalisme temporel. D'autre part, les règles de survie des robots sont en général programmées de façon liée à l'architecture, et destinées à réagir chacune à un certain type de condition : elles ne constituent pas un cadre général de spécification.

5.3.1 Règles simples

La spécification de l'exécution d'un plan R en réaction à une condition C sera notée :

dès-que(C,R)

On veut que la réaction ait lieu dès que la condition est satisfaite sur un intervalle de vérité I_c : si elle ne l'est pas sur le début de l'intervalle I, c'est-à-dire si I_c (d f oi) I, alors sa négation l'est sur un intervalle I_{nc} tel que I_{nc} (o s) I. Pour ne pas permettre que C soit satisfaite sur un intervalle placé *dans* I et *avant* I_c sans qu'il y ait de réaction, il faut spécifier que : I_{nc} m I_c. Cette situation se décrit temporellement par :

$$
\begin{array}{lll}
(\exists I_c)(& I_c\ (d\ f\ oi)\ I \wedge vrai(I_c, C) & \\
\wedge & (\exists I_{nc}) & (I_{nc}\ (o\ s)\ I \wedge I_{nc}\ m\ I_c \\
 & & \wedge\ vrai(I_{nc}, non(C))\ ) \\
\wedge & (\exists I_R) & (I_R\ (s\ si\ =)\ I_c \\
 & & \wedge I_R\ f\ I \wedge\ exec(I_R, R)\ )\ )
\end{array}
$$

On pourra remarquer que de $I_{nc}\ (o\ s)\ I \wedge I_{nc}\ m\ I_c$ on peut déduire, par les tables de transitivité, que $I_c\ (d\ f\ oi)\ I$; la formulation est donc redondante, mais c'est en faveur du caractère explicite de la description.

Dans un autre cas d'exécution, l'intervalle I d'exécution de la règle est celui du plan de réaction R : ce cas correspond à la satisfaction de la condition C sur le début de la règle, c'est-à-dire sur I_c tel que $I_c\ (o\ s\ fi\ di\ si\ =)\ I$. Ceci se décrit temporellement par :

$$
\begin{array}{ll}
(\exists I_c)(& I_c\ (o\ s\ fi\ di\ si\ =)\ I \\
 & \wedge\ vrai(I_c, C) \wedge\ exec(I, R)\ )
\end{array}
$$

Enfin, dans le cas où la condition de réaction n'est pas satisfaite sur l'intervalle I de la règle, le plan de réaction n'est pas exécuté :

$$(\exists I_c)(\ I\ dans I_c \wedge vrai(I_c, non(C))\ )$$

La définition temporelle de la primitive *dès-que* est alors :

Définition 16 :

$$
\begin{array}{lll}
\text{exec}(I, \text{dès-que}(C, R)) \Longleftrightarrow & (\exists I_c)(& (\ I_c\ (o\ s\ fi\ di\ si\ =)\ I \\
 & & \quad \wedge\ \text{vrai}(I_c, C) \wedge\ \text{exec}(I, R)\ ) \\
 & \vee & (\ I_c\ (d\ f\ oi)\ I \wedge \text{vrai}(I_c, C) \\
 & & \wedge\ (\exists I_{nc})(I_{nc}\ (o\ s)\ I \wedge I_{nc}\ m\ I_c \\
 & & \wedge\ \text{vrai}(I_{nc}, non(C))\ ) \\
 & & \wedge\ (\exists I_R)(I_R\ (s\ si\ =)\ I_c \wedge I_R\ f\ I \\
 & & \wedge\ \text{exec}(I_R, R)\ )\ ) \\
 & \vee & (\ I\ dans I_c \wedge vrai(I_c, non(C))\ ) \\
 &) &
\end{array}
$$

On peut remarquer, pour faire un lien avec la section 5.2, que la primitive *attente(C)* a le même comportement que *dès-que(C,rien)*.

5.3.2 Règles répétitives

Une extension possible de cette primitive est de la rendre répétitive, c'est-à-dire de permettre de spécifier que la réaction est exécutée à chaque fois que la condition sera satisfaite. On notera cette nouvelle primitive :

à-chaque-fois-que(C,R) ou *acfq(C,R)*

Son exécution sur un intervalle I s'envisage comme suit : à chaque fois que la condition C sera vérifiée sur un intervalle I_C commençant dans I, c'est-à-dire que $I_c\ (d\ f\ oi)\ I$, on a une exécution immédiate du plan de réaction R sur un intervalle I_R qui fait partie de l'intervalle de la règle I, c'est-à-dire que $(I_R\ (s\ si\ =)\ I_c \wedge I_R\ dans\ I)$.

On distingue, comme pour la primitive de réaction *dès-que*, le cas où C est satisfaite sur le début de I, c'est-à-dire $I_c\ (o\ s\ fi\ di\ si\ =)\ I$ et $I_R\ s\ I$. Enfin, une autre exécution possible de *acfq(C,R)* est celle où la condition C n'est pas vérifiée dans I, et alors la réaction R n'est pas exécutée.

La définition temporelle de *acfq(C,R)* est :

Définition 17 :

$$
\begin{array}{lll}
\text{exec}(I, \text{acfq}(C, R)) \Longleftrightarrow & (\forall I_c)(\exists I_R)(& \text{vrai}(I_c, C) \wedge \text{exec}(I_R, R) \wedge \\
 & & (\ (I_c\ (d\ f\ oi)\ I \wedge I_R\ (s\ si\ =)\ I_c \\
 & & \quad \wedge I_R\ dans\ I) \\
 & & \quad \vee (I_c(o\ s\ fi\ di\ si\ =) I \wedge I_R\ s\ I)\ ) \\
 &) & \\
 & \vee & \\
 & (\exists I_{nc})(& I\ dans I_{nc} \wedge \text{vrai}(I_{nc}, non(C))\ )
\end{array}
$$

On remarque que cette définition autorise le "cumul" des exécutions de R dans les cas où C a un intervalle de vérité chevauché par un intervalle d'exécution de R. Or on peut vouloir éviter cette situation, en imposant un séquencement des réactions ; on se donne pour cela une autre primitive :

afcq2(C,R)

définie temporellement par une itération sur une règle non-répétitive :

Définition 18 :
action-c(acfq2(C,R), tant-que(vrai, dès-que(C,R)))

5.4 Utilisation des règles

5.4.1 Association d'une règle à un plan

La primitive *dès-que* a parmi ses caractéristiques de pouvoir donner lieu à des exécutions pour lesquelles l'intervalle correspondant n'est pas déterminé : en particulier les cas où la condition de réaction n'est pas rencontrée. Au moment d'envisager son utilisation dans le cadre de la structure de contrôle d'un plan englobant, on peut faire le lien avec les actions à durée indéfinie, vues en section 5.2, dont l'intervalle n'était pas déterminé par la définition temporelle. Toutefois, l'intervalle d'exécution d'une règle est plus contraint que celui d'une action à durée indéfinie, de par l'exécution du plan de réaction, ce qui introduira des différences dans la définition des primitives.

On spécifie qu'une règle D s'exécute *aussi longtemps que* s'exécute un sous-plan P en parallèle en les associant dans une primitive notée :

assoc(P, D)

où D est de la forme *dès-que(C,R)* et R est le plan de réaction.

Son interprétation intuitive est que, durant l'exécution du sous-plan P, la condition C donnera lieu à la réaction R ; une fois le plan terminé, la règle cessera d'être exécutée : on représente ainsi sa "désactivation".

Dans une exécution où la règle D : *dès-que (C,R)* s'exécute sur l'intervalle I_D qui termine avant l'intervalle I_P du plan P : la primitive *assoc(P,D)* s'exécute sur I tel que $(I = I_P \wedge I_D\ (s =)\ I)$.

Une autre exécution possible est celle où le plan P s'exécute sur I_P qui termine avant I_D. Dans ce cas, la primitive *assoc* voit son intervalle I déterminé par celui de la règle : $(I = I_D \wedge I_P\ (s =)\ I)$.

Une troisième exécution possible est celle où la condition C n'est jamais satisfaite durant l'intervalle d'exécution de P, dont l'intervalle détermine alors celui de la structure englobante *assoc* , comme celui de la primitive D : *dès-que(C.R)* qui lui est associée : $(I_P = I \wedge I_D = I)$.

La définition temporelle de la primitive *assoc(P, dès-que(C,R))* est alors :

Définition 19 :

$$\begin{aligned}\text{exec(I, assoc(P, D))} \Longleftrightarrow (\exists I_P, I_D)(\quad & \text{exec}(I_P, P) \wedge \text{exec}(I_D, \text{D}) \wedge \\ & ((I = I_P \wedge I_D\ s\ I) \\ & \vee (I = I_D \wedge I_P\ s\ I) \\ & \vee (I = I_P \wedge I_D = I)))\end{aligned}$$

Des primitives peuvent être définies pour associer à un plan une règle répétitive, un ensemble de règles simples, ou un ensemble de règles répétitives [Rutten 90].

5.4.2 Association d'une règle à une condition

Une autre façon d'inclure une règle dans une structure de contrôle est de spécifier qu'elle doit être exécutée tant qu'une condition est satisfaite dans l'environnement extérieur.

Dans le cas d'une règle D de la forme *dès-que(C_R,R)*, on se donne pour cela la primitive :

tq-c-r(C, D)

dont l'interprétation est que la règle D est exécutée tant que la condition C est vérifiée dans l'environnement ; c'est-à-dire que D est désactivée quand C n'est plus satisfaite.

Comme pour l'association à un plan, on distingue trois cas suivant l'exécution de la règle D :

- soit elle s'exécute sur un intervalle I_D tel que I_D termine avant I_C, la réaction ayant été déclenchée et la règle exécutée sur I_D : la primitive *tq-c-r(C, D)* s'exécute alors sur l'intervalle I tel que $(I\ (f\ =)\ I_C\ \wedge\ I_D\ (s\ =)\ I)$.
- soit c'est I_C qui termine avant I_D, dans le cas où le plan de réaction s'exécute sur un intervalle chevauchant I_{nc}. Alors I est déterminé par l'intervalle de la règle : $(I\ =\ I_D\ \wedge\ I_C\ (o\ s\ =)\ I)$.
- d'autre part, si la condition C n'est pas vérifiée sur I, alors la règle n'est pas exécutée.

On a donc la définition :

Définition 20 :

$$\begin{aligned}\text{exec(I, tq-c-r(C,D))} \Longleftrightarrow (\exists I_C, I_{nc}, I_D)(\quad & \text{vrai}(I_C, C) \wedge \text{vrai}(I_{nc}, non(C)) \wedge I_C\ m\ I_{nc} \\ & \wedge \text{exec}(I_D, D) \wedge \\ & (\ (I\ (f\ =)\ I_C\ \wedge\ I_D\ (s\ =)\ I) \\ & \vee (I\ =\ I_D\ \wedge\ I_C\ (o\ s\ =)\ I)\ ) \\ &) \\ \vee (\exists I_{nc})(\quad & \text{vrai}(I_{nc}, non(C)) \wedge I\ dans I_{nc} \wedge \text{dure}(I, 0)\ )\end{aligned}$$

De même que précédemment, on peut définir le même type de primitive pour les règles répétitives, les ensembles de règles simples, et les ensembles de règles répétitives.

5.5 Communication avec l'opérateur

Le modèle temporel de plans d'actions vise une application en télérobotique, et dans ce cadre l'opérateur doit pouvoir disposer de moyens pour spécifier son intervention dans le déroulement de l'exécution des plans qu'il compose. Suivant l'approche de représentation logique et temporelle suivie ici, les interactions entre l'exécution du plan et l'opérateur concernent essentiellement le choix parmi plusieurs possibilités d'exécution établies à l'avance. Ce choix peut prendre la forme de l'alternative entre plusieurs sous-plans, ou de l'exécution facultative d'un sous-plan.

Alternative La primitive d'alternative entre deux sous-plan est notée : *alt($Plan_1$, $Plan_2$)* et sa signification intuitive est que l'un des deux plans sera choisi par l'opérateur et exécuté. Les sous-plans de l'alternative peuvent être vus par exemple comme deux façons d'arriver à un même résultat (dans un cadre de téléopération, on peut citer l'exemple d'alternative entre une exécution d'une action en mode automatique ou en mode de télémanipulation par l'opérateur [Gravez 88]) ou bien plus généralement entre deux options différentes. On l'intègre au modèle par la définition :

Définition 21 :

$$\text{exec(I,alt}(P_1, P_2) \Longleftrightarrow \text{exec(I,}P_1) \vee \text{exec(I,}P_2)$$

De la même façon qu'au chapitre 4 on a généralisé la conditionnelle en cas, on peut ici définir une alternative sur un ensemble de sous-plans, notée : *alt-ens(EP)* dont la signification est qu'un et un seul parmi les sous-plans de l'ensemble EP sera choisi par l'opérateur pour être exécuté. On l'intègre au modèle temporel par la définition :

Définition 22 :

$$\text{exec(I,alt-ens(EP)} \Longleftrightarrow (\exists! p \in EP)\ \text{exec(I,}p)$$

Une utilisation possible de l'alternative est de donner le choix entre des plans différents, mais ayant des effets globaux équivalents ; toutefois il n'y a pas de raison de faire de ce cas une contrainte sur les sous-plans de l'alternative en général.

Facultative La primitive facultative est notée : *fac(P)* et sa signification est que soit le sous-plan P est exécuté, soit il ne l'est pas. On l'intègre au modèle temporel en la définissant, au moyen de l'alternative vue en définition 21 :

Définition 23 :

exec(I,fac(P)) $\Longleftrightarrow$ exec(I,alt(P, rien))

On remarquera que cette définition est similaire à celle de la conditionnelle simple vue en section 4.3.

6 Exemple d'utilisation du modèle

6.1 Exemple de description d'environnement

Un exemple peut être donné dans un environnement de station spatiale. Nous n'en donnons ici qu'une présentation résumée pour rester bref, un présentation complète en est donnée ailleurs [Rutten 90]. Il est inspiré d'études réalisées par MATRA-ESPACE [André e.a. 89]. Sur celle-ci se trouve un système de manipulation à deux bras, *bras1* et *bras2*, tous deux *disponibles* dans la situation initiale, *bras1* étant dans la position de repos *posrepos1*, et *bras2* dans la position de repos *posrepos2*. Un *outil* est rangé à la position *posoutil*, et il est *libre*. Une *pièce* amovible est située à la position *pospiece*, et *fixée sur* un *support*. Les *communications* entre le système télérobotique spatial et le système de supervision à terre ne peuvent se faire que par période, du fait de leur interruption dûe à la trajectoire de la station spatiale.

La situation correspond à l'ensemble de faits temporels suivant, où à chaque propriété est associé un intervalle : *disponible(bras1), disponible(bras2), position(bras1, posrepos1), position(bras2, posrepos2), libre(outil), position(outil, posoutil), position(piece, pospiece), fixesur(piece, support).*

6.2 Exemples d'action

Dans l'exemple d'environnement considéré, on définit l'action, d'une durée de 1 *mn*, pour un *bras*, de *lâcher* un *objet* :

action(lacher(Bras,Obj), 1 mn,
[tenu(Obj,Bras)] ,
[tenu(Obj,Bras)] ,
[non(tenu(Obj,Bras)),
disponible(Bras)]).

Cette action a pour précondition et pour condition que l'*objet* doit être *tenu* par le *bras*, et pour effets que l'objet n'est plus tenu par le bras, et que le bras devient *disponible*.

Une action aux effets réciproques est celle de *saisir* un objet. Elle est décomposée en deux étapes, et définie par :

action-c(saisir(Bras,Obj),
seq([saisir1(Bras,Obj) ,
saisir2(Bras,Obj)]))

avec :

action(saisir1(Bras,Obj), 0,
[] ,
[disponible(Bras), position(Obj,Pos),
position(Bras,Pos)] ,
[non(disponible(Bras))])

pour laquelle les conditions sont que le bras et l'objet soient à la même position, et que le bras soit disponible, et dont l'effet est que le bras n'est plus disponible.

A l'issue de cette action l'*objet* n'est pas encore tenu ; c'est là qu'intervient la deuxième partie de l'action composée :

```
action( saisir2(Bras,Obj), 1 mn,
 [ ] ,
 [position(Obj,Pos),position(Bras,Pos)] ,
 [tenu(Obj,Bras)] )
```

Ces actions sont des exemples d'action primitive et d'action composée. Les suivantes seront présentées plus brièvement.

Une action de manipulation est celle de faire *détacher* un objet par un bras en utilisant un outil : *detacher(Bras, Outil, Obj)*. Les conditions en sont que, l'objet étant fixé sur un support, l'outil soit tenu par le bras, et que tous trois soient à la même position ; les effets sont que l'objet n'est plus fixé sur son support, et est libre.

On définit pour les mouvements une action nommée *allerà(Obj,Pos)*, qu'on décompose pour rendre compte du mouvement, qui implique dès le départ qu'on n'est plus à la position précédente, et qu'on est en mouvement entre celle-ci et la position de destination.

Une action à durée indéfinie utile est celle, pour un *bras*, de *tenir* un *objet* : il s'agit d'une action de maintenance de condition, dans le sens où elle n'a pas d'effet en tant que tel, mais doit assurer la préhension de l'*objet* par le *bras*.

```
action-dv(  tenir(Bras,Obj),
            [ ],
            [tenu(Obj,Bras)],
            [ ] ).
```

Un exemple d'utilisation de cette action dans une primitive l'associant à un sous-plan parallèle est de la faire exécuter par le *bras2* sur la *pièce* aussi longtemps que le *bras1* exécute le plan où il détache la pièce de son support, retourne ranger l'outil utilisé et revient saisir la pièce, tout ceci nécessitant son maintien par l'autre bras, pour éviter qu'elle ne parte à la dérive ou pour faciliter les opérations de manipulation. Ce sous-plan peut être redéfini comme une action composée, *détacher2*, consistant en une opération conjointe de deux bras pour détacher une pièce ; les actions préliminaires sont à exécuter par ailleurs, comme nous le verrons dans l'exemple de plan. Cette action composée a la forme suivante :

```
action-c(  detacher2(bras1, bras2, outil, pièce) ,
  a-l-q(   seq([  detacher(bras1, outil, piece),
                  allera(bras1,posoutil),
                  lacher(bras1,outil),
                  allera(bras1,pospiece),
                  saisir(bras1,piece) ])
           tenir(bras2,piece) ) )
```

6.3 Exemple de plan

On veut faire détacher la *pièce* de son *support* par le *bras 1* au moyen de l'*outil*, et avec l'assistance du *bras 2*, qui tiendra la *pièce*.

Un plan correspondant est alors:

```
seq([  par([  seq([  allera(bras1,posoutil).
                     saisir(bras1,outil),
                     allera(bras1,pospiece) ]),
              seq([  allera(bras2,pospiece),
                     saisir(bras2,piece) ])
           ]),
       detacher2(bras1, bras2, outil, pièce)
       ])
```

6.4 Disposition temporelle

Les figures 3 et 4 sont des résultats obtenus par le simulateur TEMPUS FUGIT [Rutten e.a. 89a].

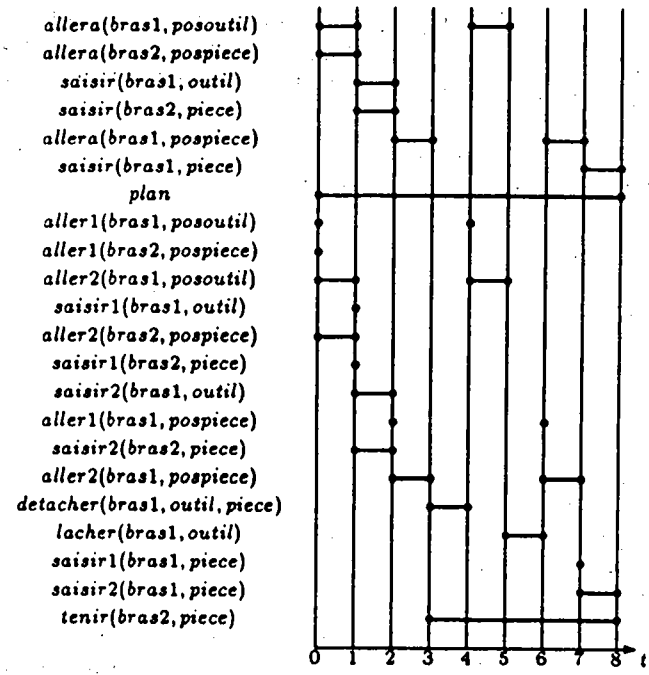


Figure 3: Exemple de plan : les actions dans le temps

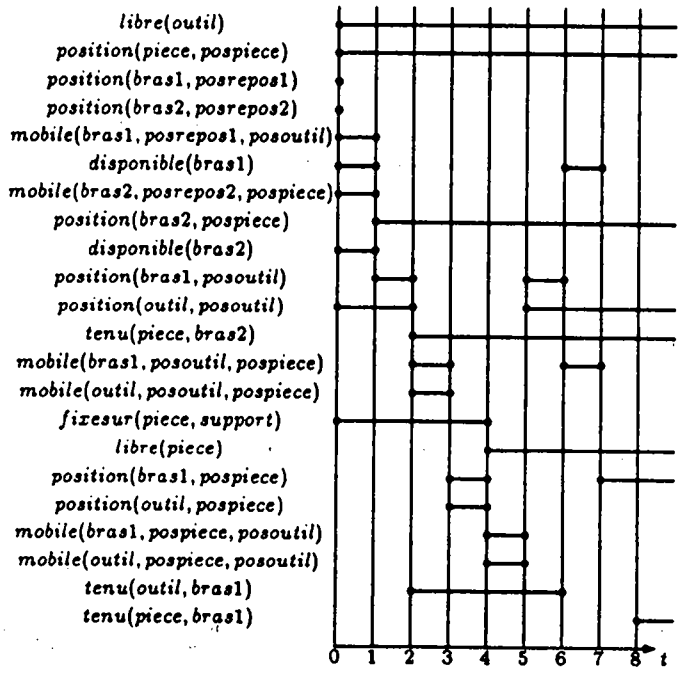


Figure 4: Exemple de plan : les faits temporels dans le temps

6.4.1 Actions et plan

La disposition des actions et du plan dans le temps est illustrée en fig. 3. On y trouve les intervalles d'exécution des actions composées, du plan, et des actions primitives, ainsi que celui de l'action à durée indéfinie *tenir*, qui est déterminé par celui du sous-plan associé dans la primitive *a-l-q*, comme on l'a vu plus tôt.

L'ordre dans lequel sont affichés les faits temporels ne correspond pas à un classement particulier : ceci nécessite un outil de visualisation plus performant.

6.4.2 Histoire de l'environnement

L'histoire de l'environnement subissant l'exécution de l'exemple de plan est donnée par les fait temporels illustrés en fig 4.

7 Conclusion

7.1 Un modèle structuré de plans d'actions

Nous avons défini un modèle logique et temporel de plans d'actions dotés d'une structure de contrôle impérative, sur la base d'un formalisme unifié inspiré de la logique à intervalles d'Allen, permettant de décrire aussi bien l'environnement et son évolution que l'arrangement temporel des plans.

Les primitives de contrôle sont définies d'abord de façon indépendante de l'environnement, leur disposition temporelle ne dépendant que de leurs composants. Ensuite, d'autres primitives sont définies comme des réactions à l'environnement, prenant en compte les intervalles de vérité de faits temporels. En relation avec le contrôle de l'exécution de plans par un robot [Rutten e.a. 90,Paoletti 91], ce modèle fournit la base d'un langage de programmation de robot au niveau tâche doté d'une définition formelle. Comme un exemple simplifié le suggère ici, le modèle a été appliqué à la simulation de plans de missions en télérobotique spatiale. concernant un robot bi-bras [André e.a. 89] : cette expérimentation a fourni un banc d'essai à une première version du modèle, et inspiré certaines de ses extensions [Rutten 90].

7.2 Extensions et perspectives

D'autres extensions et travaux futurs concernent :

- le modèle lui-même : une étude plus complète de ses éléments, de la façon dont ils peuvent être combinés, et de leurs propriétés est nécessaire pour mieux déterminer la façon dont on peut les utiliser au mieux et leur signification vis-à-vis des applications visées, et de les étendre et modifier en conséquence;
- la relation entre le modèle et des outils de génération de plans, sur la base du formalisme commun de l'action, pour fournir des fonctionnalités d'assistance à l'opérateur plus proches de l'automaticité.
- son application à la programmation au niveau tâche de systèmes robotiques, dans le cadre d'une architecture de contrôleur de robot [Borelly e.a. 90]; ceci peut etre fait en relation avec un modèle des tâches-robot [Coste-Manière 89,Espiau e.a. 90] qui utilise les concepts de systèmes réactifs et synchrones du langage temps-réel de haut-niveau ESTEREL [Berry e.a. 87].

Bibliographie

[Alami e.a. 90] R. Alami, R. Chatila, M. Ghallab, C. Laugier, J.-P. Laumond. Robotique et intelligence artificielle. In *Actes des 3$^{\text{è}}$ Journées Nationales du PRC-GDR Intelligence Artificielle*, pages 411–473, Paris-La Défense, Mars 1990.

[Allen 81] J.F. Allen. An interval-based representation of temporal knowledge. In *Proceedings of the IJCAI '81*, pages 221–226, Vancouver, August 1981.

[Allen 83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[André e.a. 89] G. André, G. Berger, A. Elfving. The Bi-Arm Servicer, a multi-mission concept and a technological model for space robotics. In *Proceedings of the 2nd European In-Orbit Operations Technology Symposium*, Toulouse, France, September 1989, ESA SP-297 (December 1989).

[Andrews e.a. 83] G. Andrews, F. Schneider. Concepts and notations for concurrent programming. *ACM Computing Surveys*, 15(1):3–43, 1983.

[Audureau e.a. 87] E. Audureau, L. Fariñas del Cerro, P. Enjalbert. Théorie de la programmation et logique temporelle. *T.S.I. Technique et Science Informatiques*,6(6):527–540, 1987(Première partie), et 7(2):181–200, 1988 (Seconde partie).

[Benveniste e.a. 89] A. Benveniste, P. Le Guernic, C. Jacquemot. *The SIGNAL software environment for real-time system specification, design, and implementation.* Publication Interne IRISA, Rennes, n° 490, Septembre 1989.

[Berry e.a. 87] G. Berry, P. Couronné, G. Gonthier. Programmation synchrone des systèmes réactifs : le langage ESTEREL. *T.S.I. Technique et Science Informatiques*, 6(4):305–316, 1987.

[Bestougeff e.a. 89] H. Bestougeff, G. Ligozat. *Outils logiques pour le traitement du temps. De la linguistique à l'intelligence artificielle.* Masson, collection Etudes et Recherches en Informatique, Paris, 1989.

[Borelly e.a. 90] J.-J. Borelly, D. Simon. *Proposition d'architecture de contrôleur ouvert pour la robotique.* Rapport de Recherche INRIA no. 1304, Octobre 1990.

[Caspi e.a. 87] P. Caspi, D. Pilaud, N. Halbwachs, J. Plaice. Lustre, a declarative language for programming synchronous systems. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, POPL '87*, Munich, Germany, pages 178–188, 1987.

[Coste-Manière 89] E. Coste-Manière. *Utilisation d'Esterel dans un contexte asynchrone : une application robotique.* Rapport de Recherche INRIA no. 1139, Décembre 1989.

[DARPA 87] DARPA Santa Cruz Workshop on Planning, W. Swartout *ed.*, *AI Magazine*, 9(2):115–131, Summer 1988.

[Dubois e.a. 91] E. Dubois, J. Hagelstein, A. Rifaut. A formal language for the requirements engineering of computer systems. To appear in *From natural language processing to logic for expert systems*, A. Thayse (ed.), Wiley, 1991.

[Espiau e.a. 90] B. Espiau, E. Coste-Manière. A synchronous approach for control sequencing in robotic applications. In *Proceedings of the IEEE Workshop on Intelligent Motion Control.* Istambul, August 1990.

[Filman e.a. 84] R. Filman, D. Friedman. *Coordinated computing; tools and techniques for distributed software.* McGraw-Hill, Computer Science series, New York, 1984.

[Gaspart 87] P. Gaspart. *Langages de programmation de la robotique.* Hermes, Paris, 1987.

[Ghallab 89] M. Ghallab. Représentation et gestion de relations temporelles. In *Actes du 7è Congrès AFCET-INRIA de Reconnaissance des Formes et Intelligence Artificielle, RFIA '89*, pages 3–20, Paris, 27 Novembre - 1er Décembre 1989.

[Gravez 88] P. Gravez. *Etude d'un système de supervision pour la téléopération assistée par ordinateur.* Thèse de Doctorat de 3è cycle de l'Université des Sciences et Techniques de Lille, 23 Mars 1988.

[Hendler e.a. 90] J. Hendler, A. Tate, M. Drummond. AI Planning: Systems and Techniques. *AI Magazine*, 11(2):61–77, Summer 1990.

[Hultman e.a. 88] J. Hultman, A. Nyberg. *Realizing action plans and response rules in a system tool for an autonomous vehicle.* Technical report n° LiTH-IDA-R-88-24, Department of Computer and Information Science, University of Linköping, Sweden, 1988.

[Hultman e.a. 89] J. Hultman, A. Nyberg, M. Svensson. *A software architecture for autonomous robots.* Technical report n° LAIC-IDA-89-TR1, Department of Computer and Information Science, University of Linköping, Sweden, 1989.

[Jauffrineau 88] C. Jauffrineau. *Contrôle d'exécution en robotique de coopération.* Publication Interne IRISA, Rennes, n° 416, Juin 1988.

[Michalowski e.a. 87] S. Michalowski, C. Crangle, L. Liang. A natural-language interface to a mobile robot. In *Proceedings of the Workshop on Space Telerobotics*, NASA, JPL Publication 87-13, vol. II, pages 381–391, Pasadena, July 1987.

[Paoletti e.a. 89] J.-C. Paoletti, L. Marcé. *Quelques outils graphiques pour la modélisation du contrôle d'exécution en robotique de coopération.* Publication Interne IRISA, Rennes, n° 475, Juin 1989.

[Paoletti 91] J.-C. Paoletti. *Spécification et sémantique opérationnelle d'un langage de contrôle d'exécution de plans d'actions pour la télérobotique.* Thèse de Doctorat de l'Université de Rennes I, 26 Mars 1991.

[Rutten e.a. 89a] E. Rutten, L. Marcé. Plan simulation using temporal logics. Rapport de Recherche INRIA no. 1095, Septembre 1989.

[Rutten e.a. 89b] E. Rutten, L. Marcé. Logiques temporelles et plans d'actions structurés. In *Actes du 7è Congrès AFCET-INRIA de Reconnaissance des Formes et Intelligence Artificielle, RFIA '89*, Paris, 27 Novembre - 1er Décembre 1989.

[Rutten 90] E.-P. Rutten. *Représentation en logique temporelle de plans d'actions dotés d'une structure de contrôle impérative; application à l'assistance à l'opérateur en téléopeération.* Thèse de Doctorat de l'Université de Rennes I, 13 Juillet 1990.

[Rutten e.a. 90] E. Rutten, J.C. Paoletti, G. André, L. Marcé. A task-level language for operator assistance in teleoperation. In *Proceedings of the International Conference on Human Machine Interaction and Artificial Intelligence in Aeronautics and Space*, Toulouse, Septembre 1990.

[Sandewall e.a. 86] E. Sandewall, R. Rönnquist. A representation of action structures. In *Proceedings of the AAAI '86*, pages 89–97, Philadelphia, August 1986.

[Scapin e.a. 89] D. Scapin, C. Pierret-Goldbreich. MAD : une méthode analytique de description des tâches. In *Actes du Colloque sur l'Ingénierie des Interfaces Homme-Machine*, pages 131–148, Sophia-Antipolis, 24-26 Mai 1989.

[Schroeder-Trovato e.a. 87] K. Schroeder-Trovato, W. Schreiner. LABICS: a language for distributed, hierarchical, and dynamically reconfigurable control system. In *Proceedings of the 1987 IEEE Workshop on Languages for Automation*, pages 43–47, Vienna, Aug. 1987.

[Shoham 86] Y. Shoham. *Reasoning about change : time and causation from the standpoint of artificial intelligence.* Ph.D. Thesis, Yale University, Computer Science Department, December 1986.

[Voltz 88] R. Voltz. Report of the robot programming languages working group. NATO Workshop on Robot Programming Languages. *IEEE Journal of Robotics and Automation*, RA 4(1):86–90, February 1988.

Table des matières

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249 - 6399